



8장. 프로퍼티



8.1. 코틀린에서 프로퍼티 이용

8.1.1. 프로퍼티란?

- 코틀린에서는 클래스의 변수를 프로퍼티(Property)라 부른다.
- var, val로 선언되는 변수들이 프로퍼티 이다.
- getter/setter(accessor)함수가 자동으로 내장

```
class User {  
    var name: String = "kkang"  
    val age: Int = 20  
}  
  
fun main(args: Array<String>) {  
    val user=User()  
  
    user.name="kim"  
    println("name : ${user.name}")  
    println("age : ${user.age}")  
}
```

8.1. 코틀린에서 프로퍼티 이용

```
class User {  
    var name: String = "kkang"  
        get() = field  
        set(value) {field=value}  
  
    val age: Int = 20  
        get() = field  
}  
  
fun main(args: Array<String>) {  
    val user=User()  
  
    user.name="kim"  
    println("name : ${user.name}")  
    println("age : ${user.age}")  
}
```

- var로 선언한 변수는 값의 획득 및 변경이 가능함으로 get(), set() 이 추가
- val로 선언한 변수는 값의 변경이 불가능함으로 get() 만 추가
- get(), set() 의 field는 예약어로 프로퍼티에 저장된 값 자체를 지칭

8.1. 코틀린에서 프로퍼티 이용

8.1.2. 사용자 정의 프로퍼티

```
//custom property
class User1 {
    var greeting: String = "Hello"
    set(value) {
        field = "Hello" + value
    }
    get() = field.toUpperCase()

    var age: Int=0
    set(value) {
        if(value >0 ){
            field=value
        }else {
            field=0
        }
    }
}

fun main(args: Array<String>) {
    val user1=User1()
    user1.greeting="kkang"
    println(user1.greeting)
    user1.age=-1
    println("age : ${user1.age}")
}
```

8.1. 코틀린에서 프로퍼티 이용

- `get()`, `set()` 내부에서는 프로퍼티 값을 `field`로 접근한다.
- `var`의 경우는 `set()` 과 `get()` 을 모두 정의할수 있지만 `val`의 경우는 `set()` 을 정의할수 없다.
- `val`의 경우 `get()` 을 정의하였다면 초기 값을 명시하지 않아도 된다.
- `var`의 경우는 `get()` 이 정의되었다고 하더라도 초기 값이 명시되어야 한다.

```
class User2 {  
    //val 의 set() 사용시 예러  
    val name: String = "kkang"  
        get() = field.toUpperCase()  
        set(value) { field = "Hello" + value } //error  
  
    //val의 초기값 생략 가능  
    val age: Int  
        get() = 10  
  
    //var의 경우 초기값 생략 불가능  
    var phone: String //error  
        get() = "01000000"  
}
```

8.1. 코틀린에서 프로퍼티 이용

8.1.3. 주생성자와 프로퍼티

- 프로퍼티로 이용되는 것은 Top-Level 변수와 클래스 내부의 변수

```
//Top-Level Variable
var myVal: String="hello"
    get() = field.toUpperCase()
    set(value){
        field = "hello" + value
    }

class User1(val name: String){
    //Class Member Variable
    var age: Int=0
        set(value) {
            if(value>0) field=value
            else field=0
        }

    fun myFun(){
        //함수의 local variable은 프로퍼티가 아니다.
    }
    var no=0
    //        get() = field * 10//error
}
```

8.1. 코틀린에서 프로퍼티 이용

- 주생성자의 var, val 이 추가된 변수에 Custom get(), set() 이 추가 불가

```
class User2(var name: String){
    var myName: String = name
    get() = field.toUpperCase()
    set(value) {
        field="Hello"+value
    }
}

fun main(args: Array<String>) {
    val user2=User2("kkang")
    user2.name="lee"
    user2.myName="kim"
    println("name : ${user2.name}") //lee
    println("myName : ${user2.myName}") //HELLOKIM
}
```


8.1. 코틀린에서 프로퍼티 이용

```
class User3(name: String){
    var name: String = name
    get() = field.toUpperCase()
    set(value) {
        field="Hello"+value
    }
}

fun main(args: Array<String>) {
    val user3=User3("kkang")
    user3.name="kim"
    println("name : ${user3.name}")
}
```


8.2. 프로퍼티 초기화

8.2.1. 초기화 블록에서 초기화

```
class User {  
    var data: String  
    val data2: Int  
  
    init {  
        data="kkang"  
        data2=10  
    }  
}
```

8.2.2. null 허용으로 선언

```
class User {  
    val name1: String="kkang"  
    var name2: String?=null  
    val name3: String?=null  
    var age: Int? = null  
  
    constructor(name2: String, name3: String, age: Int){  
        this.name2=name2  
        this.name3=name3//error  
        this.age=age  
    }  
}
```

8.2. 프로퍼티 초기화

8.2.3. 늦은 초기화

```
class User1 {  
    lateinit var lateData: String  
}  
  
fun main(args: Array<String>) {  
    //lateinit  
    val user=User1()  
    user.lateData="hello"  
    println(user.lateData)  
}
```

- lateinit은 var의 경우에만 사용이 가능하다.
- lateinit은 클래스 바디에 선언한 프로퍼티만 사용이 가능하다. 주생성자에서는 사용할수 없다.
- lateinit 은 custom getter/setter를 사용하지 않은 프로퍼티에만 사용할수 있다.
- nullable 프로퍼티에는 사용할수 없다.
- 타입은 기초 타입이 아니어야 한다.

8.2. 프로퍼티 초기화

```
lateinit var data1: String//error

class User2(lateinit var data: String) {//error
    lateinit val data2: String//error
    lateinit var data3: String?//error
    lateinit var data4: Int//error
    lateinit var data5: String//ok
}
```

8.2. 프로퍼티 초기화

8.2.4. 초기화 미루기

- 초기화를 프로퍼티의 이용 시점으로 미룰수 있는 방법
- lazy는 일종의 실행 영역
- by lazy { } 로 선언되어 프로퍼티의 초기화를 lazy의 { } 에서 진행
- 호출 시점에 초기화 진행
- val 과 같이 사용 가능
- 클래스 바디 이외에 Top-Level 에서도 사용가능
- primary type에도 사용 가능

8.2. 프로퍼티 초기화

```
val someData: String by lazy {
    println("i am someData lazy...")
    "hello"
}
class User1 {
    val name: String by lazy {
        println("i am name lazy...")
        "kkang"
    }
    val age: Int by lazy {
        println("i am age lazy...")
        10
    }
    init {
        println("i am init...")
    }
    constructor(){
        println("i am constructor...")
    }
}
fun main(args: Array<String>) {
    //lazy 초기화
    val user1=User1()
    println("name use before...")
    println("name : ${user1.name}")
    println("name use after....")
    println("age use before...")
    println("age : ${user1.age}")
    println("age use after....")
}
```

실행결과

```
i am init...
i am constructor...
name use before...
i am name lazy...
name : kkang
name use after....
age use before...
i am age lazy...
age : 10
age use after....
```

8.3. 프로퍼티 값 변경 감지

```
class User {  
    var name: String by Delegates.observable("nonValue", {props, old, new ->  
        println("old : $old ... new : $new")  
    })  
}  
  
fun main(args: Array<String>) {  
    val user=User()  
    println(user.name)  
    user.name="kkang"  
    user.name="kim"  
    println(user.name)  
}
```

실행결과

```
nonValue  
old : nonValue ... new : kkang  
old : kkang ... new : kim  
kim
```